

## 1. Inserindo PHP no HTML

Existem quatro estilos para inserir comandos PHP:

### Primeiro

```
<?php
.....
comandos
.....
?>
```

Este é o estilo XML. De preferência use este estilo ao programar com PHP 3 e 4.

### Segundo

```
<?
.....
comandos
.....
?>
```

Este é o estilo abreviado. Ele segue o padrão de instruções SGML. Para usá-lo você precisa ativar `short_tags` no arquivo `/etc/php4/apache/php.ini`.

### Terceiro

```
<script language='php'>
.....
comandos
.....
</script>
```

Este é o estilo JavaScript.

### Quarto

```
<%
.....
comandos
.....
%>
```

Este é o estilo ASP. Para usá-lo você deve ativar `asp_tags` no arquivo `/etc/php4/apache/php.ini`.

**IMPORTANTE:** Apesar de existirem estes quatro estilos, preferencialmente use o estilo XML. Esta orientação está no arquivo `/etc/php4/apache/php.ini`.

## 2. Inserindo comentários o código

O PHP suporta comentários no estilo das linguagens C, C++ e no estilo do script shell:

```
<?php
/* comentário no
estilo da
linguagem C */

// comentário no estilo da linguagem C++

# comentário no estilo do script shell
?>
```

## 3. Exibindo a data e hora do sistema

A função [date](#) formata a data e hora do sistema. Sua sintaxe é:

```
date (STRING)
```

onde *STRING* é uma string que formata a saída. Exemplo:

```
<?php
    echo date("d/m/y"); // exhibe a data no formato DD/MM/YY
    echo ", ";
    echo date("H:i"); //exibe a hora no formato HH:MM
?>
```

O código acima produz uma saída semelhante a:

```
09/04/05, 15:40
```

#### 4. Variáveis

[Acessando variáveis vindas de um formulário](#)

[Criando suas próprias variáveis](#)

[Forçando uma variável a ser de um tipo diferente](#)

[Variável variável](#)

[Constantes](#)

[Escopo de variável](#)

[Funções que manipulam variáveis](#)

##### Acessando variáveis vindas de um formulário

Para acessar uma variável enviada por um formulário HTML você pode usar uma das seguintes sintaxes:

```
$HTTP_POST_VARS['variavel']
```

```
$_POST['variavel']
```

```
$variavel
```

A primeira construção é a indicada para uso. A segunda pode ser usada se existir na sua versão do PHP. A terceira sintaxe existe mas não é indicada para uso por motivos de segurança. Para ela funcionar você tem que ativar a opção `register_globals` no arquivo `/etc/php/apache/php.ini`.

Para facilitar seu trabalho você pode criar uma nova variável em seu formulário com um nome mais abreviado, usando uma construção deste tipo:

```
$variavel = $HTTP_POST_VARS['variavel'];
```

##### Criando suas próprias variáveis

Além das variáveis enviadas por um formulário, você pode criar suas próprias variáveis. Em PHP uma variável é criada quando você atribui valor a ela pela primeira vez e seu tipo será exatamente o tipo de dado que você armazenou nela. Se você armazenar dados de outro tipo, a variável assumirá o novo tipo.

Os tipos de dados em PHP são:

- integer
- double
- string
- boolean
- array

- object

A sintaxe para criar uma variável é:

```
$variavel = valor
```

### Forçando uma variável a ser de um tipo diferente

Você pode forçar uma variável a ser de um tipo diferente usando uma coerção de tipo:

```
$nota = 3;
$preco = (double)$nota;
```

No exemplo acima, nota foi criada como integer e preco recebeu o valor de nota como um double

### Variável variável

O PHP ainda fornece o tipo variável variável. Este tipo permite alterar dinamicamente o nome de uma variável.

```
$valor = 3;
$variavel1 = 'valor';
```

assim podemos usar o nome variavel1 no lugar de valor.

```
$$variavel1 = 5;
```

o comando acima é o mesmo que

```
$valor = 5;
```

### Constantes

PHP também permite declarar constantes, para isto use a sintaxe:

```
define('VARIÁVEL', VALOR);
```

Exemplo:

```
define('preco', 10.50);
```

Existem várias constantes pré-definidas pelo PHP. Para visualizá-las clique [aqui](#).

### Escopo de variável

Quanto ao escopo de variável, PHP possui os seguintes:

- variáveis superglobais: são pré-definidas e visíveis em qualquer lugar do script
- variáveis globais declaradas em um script: visíveis por todo script exceto dentro de funções
- variáveis de função: são visíveis apenas dentro da função
- variáveis utilizadas dentro da função, mas que são declaradas como globais: refere-se a variável global

### Funções que manipulam variáveis

- `gettype(variavel)` : retorna o tipo de uma variável
- `settype(variavel, tipo)` : muda o tipo de uma variável
- `is_array(variavel)` : retorna true se a variável é um array
- `is_double(variavel), is_float(variavel), is_real(variavel)` : retornam true se a variável é um número de ponto flutuante
- `is_string(variavel)` : retorna true se a variável é uma string
- `is_object(variavel)` : retorna true se a variável é um objeto
- `isset(variavel)` : retorna true se a variável existe
- `unset(variavel)` : apaga uma variável e retorna true
- `empty(variavel)` : verifica se uma variável existe e tem valor não zero e não vazio
- `intval(variavel)` : retorna o valor da variável convertido para inteiro
- `doubleval(variavel)` : retorna o valor da variável convertido para double
- `strval(variavel)` : retorna o valor da variável convertido para string

## 5. Operadores

Basicamente, os operadores usados em PHP são os mesmos usados na linguagem C. Adicionalmente temos:

- o operador de supressão de erro: @
- o operador de concatenação de string: . (Exemplo: 'Nada resiste.' a '.' um bom papiro')
- o operador relacional "idêntico a": ===
- o operador lógico de negação: !
- o operador de referência (ponteiro): &

## 6. Estruturas de controle

As estruturas de controle usadas em PHP são basicamente as da linguagem C. Abaixo segue uma rápida revisão:

```
// desvio condicional
if (CONDIÇÃO)
{
    COMANDOS;
}
```

```
if (CONDIÇÃO)
{
    COMANDOS1;
}
else
{
    COMANDOS2;
}
```

```
// múltipla escolha
```

```
if (CONDIÇÃO1)
    COMANDOS1;
elseif (CONDIÇÃO2)
    COMANDOS2;
.....
elseif (CONDIÇÃOn)
    COMANDOSn;
```

```
switch (VARIÁVEL)
{
    case 'VALOR1':
        COMANDOS1;
        break;
    case 'VALOR2';
```

```

    COMANDOS2;
    break;
case 'VALORn':
    COMANDOSn;
    break;
default:
    COMANDOS;
    break;
}

// laços (loops)
while (CONDIÇÃO)
{
    COMANDOS;
}

for(VARIÁVEL=INÍCIO; CONDIÇÃO; INCREMENTO)
{
    COMANDOS;
}

do
{
    COMANDOS;
}
while (CONDIÇÃO);

// para interromper um laço
break => para a execução do laço
continue => pula para a próxima iteração do laço
exit => sai do script

```

## 7. Manipulando arquivos

[Abrindo um arquivo](#)

[Gravando num arquivo](#)

[Fechando um arquivo](#)

[Lendo um arquivo](#)

[Lendo um caracter por vez](#)

[Verificando se um arquivo existe](#)

[Verificando o tamanho do arquivo](#)

[Apagando um arquivo](#)

[Direcionando o ponteiro para o início do arquivo](#)

[Verificando quantos bytes foram lidos](#)

[Movendo o ponteiro pelo arquivo](#)

[Bloqueando um arquivo](#)

### Abrindo um arquivo

Para abrir um arquivo usa-se a função `fopen`:

```
$arquivo = fopen(PATH, MODO)
```

onde *arquivo* é uma variável que armazenará o ponteiro do arquivo, *PATH* é o caminho do arquivo e *MODO* pode ser:

```

r - leitura
r+ - leitura e gravação
w - gravação. Se o arquivo existir será substituído
w+ - gravação. Se o arquivo existir será substituído
a - gravação. Acrescenta ao fim do arquivo
a+ - gravação. Acrescenta ao fim do arquivo
b - binário. Utilizado em sistemas que diferenciam arquivos de texto
    de binários (windows)

```

Lembre-se de testar se o arquivo foi aberto sem problemas. Você pode construir algo como:

```

@ $arquivo = fopen("/var/www/pedido.txt", 'a');
# verificando se o arquivo foi aberto sem problemas
if (!$arquivo)
{
    echo "<p>O pedido não pode ser processado agora. Tente
mais tarde.</p>";
    exit;
}

```

Observe a utilização do operador de supressão de erro para evitar as mensagens de erro padrão do PHP.

### Gravando num arquivo

Para gravar num arquivo aberto use as funções `fwrite()` e `fputs()`. `fputs()` é um alias para `fwrite()`. A sintaxe de `fwrite()` é:

```
fwrite (ARQUIVO, STRING, COMPRIMENTO)
```

onde *ARQUIVO* é um ponteiro para um arquivo aberto com `fopen()`, *STRING* é a string a ser gravada no arquivo e *COMPRIMENTO* é o número de bytes a gravar. Se este parâmetro for fornecido, `fwrite` gravará a string até o ou alcançar seu final ou ter gravado *COMPRIMENTO* bytes.

### Fechando um arquivo

Para fechar um arquivo use:

```
fclose(ARQUIVO)
```

sendo *ARQUIVO* um ponteiro para arquivo aberto com `fopen()`.

### Lendo um arquivo

Para ler um arquivo usa-se uma das seguintes funções:

```

fgets()
fgetss()
fgetcsv()
readfile()
fpassthru()
file()
file_get_contents()
fread()

```

`fgets()` é usada para ler uma linha do arquivo e sua sintaxe é:

```
fgets(ARQUIVO, TAMANHO)
```

onde *ARQUIVO* é um ponteiro para um arquivo aberto com `fopen()` e *TAMANHO* é o tamanho da linha em bytes. Ela lê a linha até encontrar um caractere de nova linha (`/n`), encontrar um fim de arquivo (EOF) ou ter lido *TAMANHO* - 1 bytes. Para ler todas as linhas de um arquivo você deve fornecer um código de loop semelhante a:

```
while(!feof($arquivo))
{
    $linha = fgets($arquivo, 999);
    echo $linha. '<br>';
}
```

observe a utilização da função `feof()`. Ela verifica se o fim o arquivo foi atingido.

`fgetss()` é parecida com `fgets()`, a diferença é que `fgetss()` retira as tags HTML. Isto é interessante quando você está lendo um arquivo escrito por outra pessoa ou lendo uma entrada do usuário. Uma tag HTML mal colocada no código pode alterar a aparência da página. A sintaxe de `fgetss()` é:

```
fgetss(ARQUIVO, TAMANHO, TAGS)
```

onde *TAGS* são as tags HTML permitidas.

`fgetcsv()` é utilizada para dividir linhas do arquivo quando você usar um caracter separador para separar os itens ou campos do registro. Ela é interessante quando trabalhando com arquivos gerados por aplicações que criam arquivos de dados separados por um caracter padrão, como é o caso das planilhas. O resultado é retornado num array. A sintaxe de `fgetcsv()` é:

```
fgetcsv(ARQUIVO, TAMANHO, SEPARADOR, CERCADOR)
```

onde *SEPARADOR* é o caractere separador (o padrão é a vírgula) e *CERCADOR* é o caractere que envolve os campos (o padrão é o caractere de aspas duplas `"`).

Para ler todo o arquivo de uma vez use `readfile()`

```
readfile(ARQUIVO)
```

*ARQUIVO* é o caminho do arquivo a ser lido. `readfile()` retorna o número de bytes lidos e exibe o arquivo na saída padrão.

Você também pode ler todo o arquivo usando `fpassthru()`

```
fpassthru(ARQUIVO)
```

neste caso, *ARQUIVO* é um ponteiro aberto anteriormente com `fopen()`. A função lê do ponteiro até o final do arquivo e, então, fecha o arquivo. Sendo assim, você pode usar `fpassthru()` para ler todo o arquivo ou uma parte dele. Ela retorna `true` ou `false` dependendo ou não do sucesso na leitura do arquivo.

`file()` é parecida com `readfile()`

```
file(ARQUIVO)
```

onde *ARQUIVO* é o caminho do arquivo a ser lido. Ela lê todo o arquivo e o retorna no formato de um array. Cada linha é um elemento do array.

De forma semelhante, `file_get_contents(ARQUIVO)` lê todo o arquivo passado, porém o retorna como uma string.

`fread()` lê uma quantidade específica de bytes do arquivo:

```
fread(ARQUIVO, BYTES)
```

onde *ARQUIVO* é um ponteiro aberto com `fopen()` e *BYTES* é o número de bytes a ser lido. A função lê até o número dado de bytes ou até o fim do arquivo, o que ocorrer primeiro.

### Lendo um caractere por vez

Para ler um caractere de cada vez use `fgetc()`

```
fgetc(ARQUIVO)
```

onde *ARQUIVO* é um ponteiro de arquivo aberto com `fopen()`. Normalmente não é interessante ler todo o arquivo utilizando `fgetc()` pois ela retorna também o caractere de fim de arquivo (EOF). É melhor ler todo o arquivo usando outra função, porém, se você quiser ler todo um arquivo e exibí-lo numa página HTML, você pode usar um trecho de código parecido com este:

```
while(!feof($arquivo))
{
    $character = fgetc($arquivo);
    if(!feof($arquivo))
        echo ($character=="\n"?'<br>':$character);
}
```

observe o segundo teste de fim de arquivo, ele é necessário pois, como foi dito acima, `fgetc()` retorna também o caractere de fim de arquivo. Observe também a utilização da instrução condicional (CONDIÇÃO ? VERDADEIRO : FALSO). Ela foi usada para emitir uma quebra de linha no HTML cada vez que o caracter de nova linha (\n) for lido.

### Verificando se um arquivo existe

```
file_exists(ARQUIVO)
```

onde *ARQUIVO* é o caminho do arquivo a ser verificado.

### Verificando o tamanho do arquivo

```
filesize(ARQUIVO)
```

onde *ARQUIVO* é o caminho do arquivo. O tamanho do arquivo é retornado em bytes.

### Apagando um arquivo

```
unlink(ARQUIVO)
```

onde *ARQUIVO* é o caminho do arquivo.

### Direcionando o ponteiro para o início do arquivo

```
rewind(ARQUIVO)
```

onde *ARQUIVO* é um ponteiro de arquivo aberto com `fopen()`.

### Verificando quantos bytes foram lidos

```
ftell(ARQUIVO)
```

onde *ARQUIVO* é um ponteiro de arquivo aberto com `fopen()`.

### Movendo o ponteiro pelo arquivo

```
fseek(ARQUIVO, BYTES, ORIGEM)
```

onde:

- *ARQUIVO* é um ponteiro de arquivo aberto com `fopen()`
- *BYTES* é o número de bytes a ser deslocado o ponteiro
- *ORIGEM* é o ponto inicial a partir de onde o ponteiro vai movimentar-se. Pode ser *SEEK\_SET* para início do arquivo, *SEEK\_CUR* para a posição atual do ponteiro ou *SEEK\_END* para o final do arquivo.

### Bloqueando um arquivo

Sempre que um arquivo é aberto é interessante bloqueá-lo para que outro usuário não esteja acessando o arquivo ao mesmo tempo. Para isto use `flock()`

```
flock(ARQUIVO, OPERAÇÃO)
```

onde *ARQUIVO* é um ponteiro de arquivo aberto com `fopen()` e *OPERAÇÃO* é o tipo de bloqueio de acordo com a tabela abaixo:

LOCK\_SH - bloqueio de leitura. O arquivo pode ser compartilhado com outros usuários.  
 LOCK\_EX - bloqueio de gravação. O arquivo não pode ser compartilhado  
 LOCK\_UN - libera bloqueio existente  
 LOCK\_NB - impede bloqueio ao tentar adquirir um bloqueio

## 8. Arrays

[Array com índices numéricos](#)

[Array com índices associativos](#)

[Arrays bidimensionais](#)

[Arrays multidimensionais](#)

[Classificando Arrays](#)

[Outras funções de processamento de arrays](#)

### Array com índices numéricos

Para criar um array use uma construção do tipo:

```
$cores = array('verde', 'amarelo', 'azul', 'branco');
```

Para acessar os elementos do array:

```
echo "A cor do céu é $cores[2]";
```

o primeiro elemento do array tem o índice zero, assim no nosso array de exemplo:

```
$cores[0] é verde
$cores[1] é amarelo
$cores[2] é azul
$cores[3] é branco
```

Assim como as variáveis, os arrays são criados na primeira vez que você os utiliza. O nosso array exemplo poderia ser criado também assim:

```
$cores[0] = "verde";
$cores[1] = "amarelo";
$cores[2] = "azul";
$cores[3] = "branco";
```

a primeira linha cria o array e as outras acrescentam elementos.

Como o array tem índices numéricos, podemos acessar os elementos com um loop do tipo:

```
for($indice=0;$indice<4;$indice++)
    echo "$cores[$indice] ";
```

Também podemos usar um loop assim:

```
foreach ($cores as $cor)
    echo $cor.' ';
```

### Array com índices associativos

O array associativo usa strings como índice. Para criar um array associativo você pode usar uma construção do tipo:

```
$notas = array('joão'=>8,'maria'=>9,'josé'=>6);
```

Para acessar os elementos:

```
echo $notas['joão'];
echo $notas['maria'];
echo $notas['josé'];
```

O array associativo também pode ser criado assim:

```
$notas = array('joão'=>8);
$notas['maria'] = 9;
$notas['josé'] = 6;
```

A primeira linha cria o array e as outras acrescentam elementos.

Ainda podemos criar o array assim:

```
$notas ['joão'] = 8;
$notas['maria'] = 9;
$notas['josé'] = 6;
```

Para acessar os elementos de um array associativo devemos usar um loop assim:

```
foreach ($notas as $nota_aluno)
    echo $nota_aluno.' ';
```

observe, porém, que este loop só exibe os valores dos elementos e não seus índices. Para exibir também os índices devemos modificar o loop para:

```
foreach ($notas as $aluno => $nota)
    echo $aluno.' - '.$nota.'  
';
```

Outra maneira seria:

```
while($elemento = each($notas))
{
```

```

echo $elemento['key'];
echo ' - ';
echo $elemento['value'];
echo '<br>';
}

```

No código acima, a variável `$elemento` é um array. `key` e `0` contém o índice do elemento atual do array `$notas` e, `value` e `1` contém o valor do referido elemento. Embora não faça diferença nenhuma qual delas você utiliza, é interessante usar localizações nomeadas (`key` e `value`) em vez de numeradas (`0` e `1`).

Há, ainda, uma construção mais limpa. A função `list()` pode ser usada para dividir o array em vários valores:

```
list$($aluno,$nota) = each($notas);
```

Podemos usar o seguinte loop para acessar todo o array:

```
while(list$($aluno,$nota) = each($notas))
  echo "$aluno - $nota<br>";
```

**OBSERVAÇÃO:** Se, ao usar `each()`, quisermos utilizar o array duas vezes, teremos que apontar novamente para o início do array usando a função `reset()`.

```
reset($notas);
while(list$($aluno,$nota) = each($notas))
  echo "$aluno - $nota<br>";
```

### Arrays bidimensionais

Código	Descrição	Quantidade
A4	Papel A4	500
CAN	Caneta esferográfica	100
BOR	Borracha	50

Um array pode conter outro array. Isto nos permite criar arrays bidimensionais. Pense num array de duas dimensões como sendo uma matriz como a tabela mostrada acima com linhas e colunas. Cada linha representa um produto e cada coluna é um atributo ou campo do produto. Para expressarmos a tabela acima por meio de um array faríamos assim:

```
$produtos = array ( array('A4','Papel A4','500'),
                    array('CAN','Caneta esferográfica','100'),
                    array('BOR','Borracha','50'));
```

Para exibir os elementos deste array podemos usar este código:

```
echo $produtos[0][0].' - '.$produtos[0][1].' - '.$produtos[0][2].'  
';
echo $produtos[1][0].' - '.$produtos[1][1].' - '.$produtos[1][2].'  
';
echo $produtos[2][0].' - '.$produtos[2][1].' - '.$produtos[2][2].'  
';
```

Podemos também usar dois loops `for`, um dentro do outro:

```
for($linha = 0;$linha < 3;$linha++)
{
  for($coluna = 0;$coluna < 3;$coluna++)
  {
    echo "$produtos[$linha][$coluna] ";
  }
  echo "<br>";
}
```

Você também pode usar nomes como índices, ou seja, usar arrays bidimensionais associativos:

```
$produtos = array (array(Codigo => 'A4',
                        Descricao => 'Papel A4',
                        Quantidade => '500'),
                  (array(Codigo => 'CAN',
                        Descricao => 'Caneta esferográfica',
                        Quantidade => '100'),
                  (array(Codigo => 'BOR',
                        Descricao => 'Borracha',
                        Quantidade => '50')));
```

Para acessar os elementos deste array bidimensional associativo, poderíamos fazer assim:

```
for($linha = 0;$linha < 3;$linha++)
{
    echo "$produtos[$linha]['Codigo'] $produtos[$linha]['Descricao']
        $produtos[$linha][Quantidade]<br>";
}
```

ou assim:

```
for($linha = 0;$linha < 3;$linha++)
{
    while(list($key,$value) = each($produtos[$linha]))
    {
        echo "$value ";
    }
    echo "<br>";
}
```

### Arrays multidimensionais

```
$produtos = array(array ( array('SEC1_A4','Papel A4','500'),
                          array('SEC1_CAN','Caneta','100'),
                          array('SEC1_BOR','Borracha','50')
                        ),
                  array ( array('SEC2_A4','Papel A4','200'),
                          array('SEC2_CAN','Caneta','300'),
                          array('SEC2_BOR','Borracha','150')
                        ),
                  array ( array('SEC3_A4','Papel A4','20'),
                          array('SEC3_CAN','Caneta','400'),
                          array('SEC3_BOR','Borracha','250')
                        )
                );
```

Para acessar os elementos deste array:

```
for($secao = 0;$secao < 3;$secao++)
{
    echo "Seção $secao<br>";
    for($linha = 0;$linha < 3;$linha++)
    {
        for($coluna = 0;$coluna < 3;$coluna++)
        {
            echo "$produtos[$secao][$linha][$coluna] ";
        }
        echo "<br>";
    }
}
```

### Classificando Arrays

```
sort($array); //classifica o array com índice numérico
```

classificando arrays associativos:

```
asort($array); // classifica por valores
ksort($array); // classifica por chaves
```

classificando na ordem inversa:

```
rsort($array);
arsort($array);
krsort($array);
```

classificando arrays bidimensionais

```
$produtos = array ( array('A4','Papel A4','500'),
                    array('CAN','Caneta esferográfica','100'),
                    array('BOR','Borracha','50'));
```

Para classificar o array acima pela segunda coluna do array use:

```
function compare($x, $y)
{
    if($x[1] == $y[1])
        return 0;
    else if($x[1] < $y[1])
        return -1;
    else
        return 1;
}

usort($produtos, 'compare');
```

Se você quiser classificar pela primeira coluna basta substituir o índice de x e y, que é 1, por zero. Se quiser classificar pela segunda coluna substitua o índice por 2. A função `usort()` indica que usaremos uma função definida pelo usuário para a comparação, que no nosso caso é a função `compare()`. Outras funções disponíveis são:

```
uasort($array, 'função');
uksort($array, 'função');
```

Para classificar o array bidimensional na ordem inversa basta trocar os valores -1 e 1 do `return` na função definida pelo usuário.

### Outras funções de processamento de array

```
shuffle($array); // mistura os elementos de um array
array_reverse($array); // retorna um array com os elementos na ordem
                        inversa
current($array); // retorna o elemento corrente
each($array); // retorna o par chave/valor do elemento corrente e
               avança para o próximo elemento
end($array); // aponta para o último elemento
next($array); // aponta para o próximo elemento
pos($array); // retorna o elemento atual
prev($array); // aponta para o elemento anterior
reset($array); // aponta para o primeiro elemento
array_walk($array, 'função'); // aplica a função a cada elemento
```

```
count($array); // retorna o número de elementos
sizeof($array); // retorna o número de elementos
file(arquivo); // retorna um array cujos elementos são as linhas do arquivo
```

## 9. Strings

Funções para manipulação de strings:

```
trim() => elimina espaços em branco do início e do final da string
ltrim() => elimina espaços em branco do início
chop() => elimina espaços em branco do final
nl2br() => substitui o caracter de nova linha pela tag <br>
printf() => exibe uma string formatada no navegador. Segue o padrão C
sprintf() => retorna uma string formatada
strtoupper() => converte string para letras maiúsculas
strtolower() => converte para minúsculas
ucfirst() => coloca o primeiro caracter em letra maiúscula
ucwords() => coloca cada primeira letra de cada palavra em maiúscula
addslashes() => adiciona barras invertidas a string
stripslashes() => desfaz o efeito de addslashes
explode(SEPARADOR,STRING) => divide uma string retornando as partes em um array
substr(STRING,INÍCIO,TAMANHO) => retorna uma substring
strcmp(str1,str2) => se str1=str2 retorna zero
                    se str1>str2 retorna um número positivo
                    se str2<str1 retorna um número negativo
                    diferencia maiúsculas de minúscula

strcasecmp() => idêntica a strcmp() só que não diferencia maiúsculas de
                minúsculas

strnatcmp() => idêntica a strcmp(), porém usa a comparação lógica em vez da
                lexicografica. Ou seja, ao comparar "2" e "12", strcmp()
                consideraria 2 maior que 12, já strcasecmp() ou strnatcmp()
                considerariam "12" maior

strlen() => retorna o tamanho da string
strstr(STRING,SUBSTRING) => procura uma substring dentro de uma string.
                            Retorna a string a partir da primeira ocorrência
                            da substring
stristr() => idêntica a strstr(), só que não diferencia maiúsculas de minúsculas
strrchr() => semelhante a strstr() só que retorna a string a partir da última
                ocorrência da substring

strpos() => semelhante a strstr(), só que retorna a posição numérica da
                primeira ocorrência da substring. A primeira posição é zero

strrpos() => semelhante a strpos(), só que retorna a posição da última
                ocorrência da substring. substring tem que ser um único caracter

str_replace(SUBSTRING,NOVASUBSTRING,STRING) => substitui substring em toda string

substr_replace(SUBSTRING,NOVASUBSTRING,POSIÇÃO) => substitui uma substring em
                toda a string a partir de uma posição. Se posição for um número
                positivo trata-se de um deslocamento a partir do início da string,
                caso seja negativo trata-se de um deslocamento a partir do fim
```

## 10. Expressões regulares

Funções para manipulação de expressões regulares:

```
ereg(ER,STRING,RESULTADO) => procura pela expressão regular ER em STRING e
                            armazena as correspondências encontradas no
                            array RESULTADO

eregi()=> semelhante a ereg(), porém não faz distinção entre maiúsculas e
```

minúsculas

`ereg_replace(ER,NOVA-SUBSTRING,STRING)` => procura por ER em STRING e substitui por NOVA-SUBSTRING

`eregi_repalace()` => semelhante a `ereg_replace()`, porém não faz distinção entre maiúsculas e minúsculas

`split(ER,STRING,MAXIMO)` => divide STRING nas substrings que estão separadas por ER e as retorna num array. MAXIMO é opcional e define o número máximo de itens do array

## 11. `require()` e `include()`

Use a função `require()` para reaproveitar código e tornar os sites consistentes. Exemplo:

```
<html>
<head>
<title>Jeg's</title>
</head>
<body>
<?php
    require('cabecalho.inc');
?>

// conteúdo do site

<?php
    require('rodape.inc');
?>
</body>
```

No código acima o cabeçalho e o rodapé são carregados usando a função `require()`. Os arquivos "cabecalho.inc" e "rodape.inc" tem a codificação desejada para o cabeçalho e para o rodapé. Usando esta técnica todas as páginas do site terão o mesmo cabeçalho e rodapé, conferindo mais uniformidade e reduzindo a probabilidade de erros. Sem contar que, caso ocorra uma mudança, você só precisará alterar os arquivos "cabecalho.inc" e "rodape.inc", e não todas as páginas do site.

Por convenção os arquivos utilizados neste tipo de técnica devem ter a extensão `.inc`.

Por segurança, estes arquivos devem estar fora do diretório onde os documentos web estão, pois caso um usuário tente carregá-los no navegador ele verá os arquivos em texto simples, incluindo qualquer senha que porventura exista.

*Basicamente, a função `include()` é semelhante a `require()`. A diferença é que `include()` pode retornar um valor. Porém, se sua intenção é retornar algo, é interessante avaliar se não será melhor definir uma nova função em vez de usar `include()`.*

## 12. Funções

### Declarando uma função

```
function NOME(PARÂMETROS)
{
    COMANDOS
}
```

OBSERVAÇÃO: Por convenção os nomes de função em PHP devem ser em letras minúsculas

### Chamando uma função

```
NOME ( PARÂMETROS );
```

### Onde colocar as funções

As funções pré-definidas do PHP estão disponíveis para todos os scripts, porém a função definida pelo programador só está disponível no script onde ela foi declarada. Assim, você deve criar um arquivo `.inc` com as declarações das funções usadas em sua aplicação e introduzir este arquivo no script com `require()`.

### Parâmetros opcionais

Ao declarar a função, os parâmetros que você atribuir valores serão opcionais. Exemplo:

```
function tabela($dados, $border=1, $cellpadding=2, $cellspacing=2)
{
    echo "<table border='$border' cellpadding='$cellpadding'
        ." cellspacing='$cellspacing'>";
    reset($dados);
    $valor = current($dados);
    while($valor)
    {
        echo "<tr><td>$valor</td></tr>\n";
        $valor = next($dados);
    }
    echo "</table>";
}
```

na função acima, os parâmetros `border`, `cellpadding` e `cellspacing` são opcionais. Se estes não forem passados na chamada da função, os valores padrão definidos aqui na declaração serão assumidos.

É importante observar que os parâmetros são posicionais, assim se na chamada da função nenhuma parâmetro opcional for passado tudo bem. Porém, caso o programador queira passar o parâmetro `cellspacing`, terá que passar algum valor para `border` e `cellpadding` também. É por isso também que os parâmetros opcionais devem estar depois dos obrigatórios.

### Escopo de variáveis

- **Variáveis locais** - são declaradas e visualizadas dentro da função
- **Variáveis globais** - são declaradas e visualizadas fora da função. Não são vistas dentro da função.
- **Variáveis superglobais** - são visíveis interna e externamente. São pré-definidas pelo PHP.
- A palavra-chave `global` torna uma variável local com escopo global.
- O comando `unset()` exclui uma variável.

### Como alterar o valor de um argumento passado para uma função

Se quisermos alterar o valor de uma variável passada como argumento para uma função devemos, na declaração da função, definir a passagem do argumento por referência. Para isto basta colocar um `&` antes do nome da variável. Exemplo:

```
// passagem por valor (tradicional)
function acrescenta($valor, $incremento=1)
{
    $valor = $valor + 1;
}
```

Com a declaração acima o código abaixo não alterará o valor da variável:

```
$valor = 10;
acrescenta($valor);
echo $valor;
```

\$valor será 10.

Já com esta declaração:

```
// passagem por referência (altera o valor do argumento passado)
function acrescenta(&$valor, $incremento=1)
{
    $valor = $valor + 1;
```

```
}
```

Ao executar o trecho de código:

```
$valor = 10;
acrescenta($valor);
echo $valor;
```

\$valor será 11.

### Saindo da função

Para sair da função use o comando `return`. Este comando pode ser usado apenas para o retorno, como também pode ser usado para retornar com um determinado valor.

```
return;

return $valor;
```

### Recursão

Apesar de PHP suportar funções recursivas, estas devem ser evitadas pois são lentas e utilizam mais memória que a iteração. Sempre que possível prefira a iteração.

## 13. Orientação a objetos

[Objetos](#)  
[Poliformismo](#)  
[Herança](#)  
[Criando Classes](#)  
[Construtores](#)  
[Criando Objetos](#)  
[Manipulando os atributos](#)  
[Executando operações](#)  
[Implementando herança](#)

### Objetos

Objetos são quaisquer itens manipulados pelo programa. Páginas, arquivos, formulários, itens de formulários como botões, combos, caixas de texto, etc. tudo isso pode ser considerado objeto. Os objetos possuem atributos que são propriedades ou variáveis que referem-se aos objetos e os modificam. Eles também possuem operações que são métodos, ações ou funções que o objeto pode realizar seja para modificar-se, seja para executar alguma ação externa ou até mesmo para executar alguma ação em outro objeto.

Os objetos podem ser agrupados em classes. Uma classe possui objetos que tem os mesmos atributos e operações. Assim eles se comportam de maneira semelhante. O que pode variar é o valor destes atributos e operações.

### Poliformismo

Isto significa que classes diferentes podem ter comportamentos diferentes para a mesma operação.

### Herança

A herança permite a criação de subclasses que herdaram atributos e operações da classe pai.

## Criando Classes

A sintaxe para a criação de uma classe é:

```
class NOME
{
    var $ATRIBUTO1;
    var $ATRIBUTO2;
    .....
    var $ATRIBUTOn;

    function OPERAÇÃO1();
    {
        COMANDOS1;
    }

    function OPERAÇÃO2 ($PARAMETRO1, $PARAMETRO2)
    {
        COMANDOS2;
    }
}
```

onde:

- NOME é o nome da classe
- \$ATRIBUTO. . são os atributos da classe
- OPERAÇÃO. . são as operações da classe

Em PHP todos os atributos e operações de classe são públicos.

## Construtores

Construtor é um tipo de operação especial utilizada quando o objeto é realiza tarefas úteis na inicialização, tipo configurar atributos ou criar outros objetos necessários. O construtor é declarado da mesma maneira que qualquer outra operação, porém ele deve ter o mesmo nome da classe.

```
class NOME
{
    function NOME() // construtor
    {
        .....
    }
}
```

O construtor é chamado toda vez que criamos um objeto.

## Criando Objetos

Para criar um objeto, ação também conhecida como instanciar uma classe ou criar uma instancia, usa-se a sintaxe:

```
$NOME = new CLASSE($PARAMETRO);
```

Exemplo:

```
class teste
{
    function teste($string)
    {
        echo "Testando a classe teste.<br>";
    }
}
```

```

        echo "String passada como argumento: $string <br>";
    }
}

$objeto1 = new teste('primeiro');
$objeto2 = new teste('segundo');

```

## Manipulando os atributos

Dentro da classe existe uma variável especial chamada `$this` que é usada para manipular os atributos. Você refere-se a um atributo com a sintaxe:

```
$this->ATRIBUTO
```

Exemplo:

```

class NOME
{
    var $ATRIBUTO;

    function OPERAÇÃO($PARÂMETRO)
    {
        $this->ATRIBUTO = $PARÂMETRO;
        echo $this->ATRIBUTO;
    }
}

```

Como os atributos e operações são públicos em PHP, podemos também fazer algo como:

```

class CLASSE
{
    var $ATRIBUTO;
}

$objeto = new CLASSE();
$objeto->ATRIBUTO = 'VALOR';
echo $objeto->ATRIBUTO;

```

Porém, acessar atributos de fora da classe não é interessante. Melhor seria usar o encapsulamento. Embora isto não possa ser feito em PHP, você pode criar funções para manipular os atributos:

```

class CLASSE
{
    var $ATRIBUTO;

    function pega_atributo()
    {
        return $this->ATRIBUTO;
    }

    function configura_atributo($NOVO_VALOR)
    {
        $this->ATRIBUTO = $NOVO_VALOR;
    }
}

```

Esta abordagem tem a grande vantagem de que, caso seja necessário processar ou formatar o valor do atributo para qualquer operação como por exemplo armazenar num banco de dados, isto só será feito uma vez e não em cada ocorrência do atributo em toda a aplicação. Assim reduzimos o risco de erro e o tempo de trabalho.

## Executando operações

Podemos executar operações de uma classe de modo semelhante a manipulação de atributos:

```
class CLASSE
{
    function OP1()
    {
        .....
    }

    function OP2($PARAM1, $PARAM2)
    {
        .....
    }
}

$OBJETO = new CLASS();

$OBJETO->OP1(); // executa a operação OP1

$OBJETO->OP2(papiro,10); // executa a operação OP2

$VAR1 = $OBJETO->OP1(); // executa OP1 e coloca o resultado em $VAR1
```

## Implementando herança

Para criar uma classe descendente de outra use a sintaxe:

```
class FILHO extends PAI
{
    .....
}
```

A classe filha herda todos os atributos e operações do pai. Assim, ao usar um objeto da classe filha, podemos nos referir a atributos e operações que foram definidos somente na classe pai. A recíproca não é verdadeira, ou seja, não podemos nos referir num objeto da classe pai a um atributo ou operação criado na classe filha.

Também podemos, na classe filha, modificar o valor de um atributo ou operação definido na classe pai. Se fizermos isto o atributo ou operação referente terá valores diferentes nos objetos criados a partir da classe pai e da classe filha.

PHP não suporta herança múltipla, ou seja, em PHP um pai pode ter muitos filhos, mas o filho só pode ter um pai.